Arithmetik mit Ganzzahlen

Thursday, 13 October 2022 11:10

1.) Addition

$$\begin{array}{r} 0 \\ +\,0 \\ \hline 0 \end{array} \quad \begin{array}{r} 0 \\ +\,1 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ +\,0 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ +\,1 \\ \hline 10 \end{array}$$

Die Addition zwar Binärzahlen läuft nach den obigen Regel ab.

Beispiel: $6_{dec} = 0110_b$

$3_{dec} = 0011_b$

$$\begin{array}{rr} 6_{dec} & 0110_b \\ +\,3_{dec} & +\,0011_b \\ \hline 9_{dec} & 1001_b \end{array}$$

Übertrag

→ Übung: Wählen Sie zwei positive Integer, so dass die Summe negativ wird

$$\begin{array}{r} 0 \times 7fff \\ +\,0 \times 0001 \\ \hline 0 \times 8000 \end{array}$$

2.) Subtraktion

Ein Prozessor rechnet nicht $6_{dec} - 3_{dec}$ sondern $6_{dec} + (-3)_{dec}$

Übung?

$6_{dec} = 0110_b$

$3_{dec} = 0011_b \longrightarrow -3_{dec} = 1101_b$

$$\begin{array}{rr} 6_{dec} & 0110_b \\ +\,-3_{dec} & 1101_b \\ \hline 3_{dec} & \overset{1}{\not{1}}\,0011_b \end{array}$$

Übertrag

overflow!

$$\begin{array}{r} 0110 \\ -\,0011 \end{array} \xrightarrow{\sim} 1100 \xrightarrow{+1} 1101$$

1 Komplement, 2 Komplement

$$\begin{array}{r} 0110 \\ +\,1101 \\ \hline 0011 \end{array}$$

> Haben die addierten Zahlen unterschiedliches Vorzeichen, dann wird der overflow (Überlauf) ignoriert, ansonsten wird das Overflow-Bit gesetzt

Zähler (counter)

⋮
0xfe
0xff ← Anfangswert
0x00
0x01
0x02
0x03 ← Endwert
⋮

→ Was ist die Differenz 0x03 - 0xff (über den overflow?)

0x0003 - 0xffff (int)

$$0x03 - \underbrace{0xff}_{+0x01}$$ Zweierkomplement

$0x03 + 0x01 = 0x04$ das ist korrekt, solange der Zähler nur einmal überläuft!

> Die Differenzen von unsigned Datentypen werden korrekt bestimmt auch wenn der Zähler überläuft!

3) Multiplikation mit 2er-Potenz

$3_{dec} \cdot 4_{dec} = 12_{dec}$ $\quad 0011_b \cdot 0100_b$

$$\begin{array}{r} 0011 \\ 0000 \\ +\quad 0000 \\ \hline 1100_b \end{array}$$

Die Zahl 0011_b ist um 2 Bits nach links geshiftet worden.

3 4 =

$$\begin{array}{r} 0011 \quad 0100 \\ \hline 0000 \\ 0011 \\ 0000 \\ +\quad 0000 \\ \hline 1100 \end{array}$$

$$\begin{array}{r} 112 \cdot 23 \\ 224 \\ +\ 336 \\ \hline 2576 \end{array} \qquad \begin{array}{r} 112 \quad 26 \\ 224 \\ 672 \\ \hline 2912 \end{array}$$

In C: `int a = 3;`

`Serial.println(a << 2);` ← shift left

Bit Shift nach links.

Eine tiefgreifende Frage ist nun ist

2·2 → in Hex

genauso schnell wie

(-1)·(-1) → in Hex

in der AVR Architektur.

Übung: → Versuchen Sie diese Frage mit Dokumentation ATmega328p Befehlssatz zu beantworten

→ Wie kann man x·2 noch am Computer ausrechnen lassen?

→ Messen Sie die Rechenzeit für alle 3 Operationen

223·2048

① Multiplikation

② Bitshift

c = (a b)

uns int 2^{16} 0 .. 65535

uns long 2^{32} 0 .. 4294967295

$0x0002 \cdot 0x0002$

$0xffff \cdot 0xffff$

Was ist $0xff \cdot 0xff$ $(-1)\,(-1) =$

$$\begin{array}{r} 11111111 \quad 11111111 \\ +\quad\quad\quad\quad\quad\quad\quad\quad \\ \hline 00000001 \end{array}$$

Seite 98

$x \ll 0x1$

$0x0002 \cdot 0x0002$	22 Takte
$0xffff \cdot 0xffff$	22 Takte
$0x0002 \ll 0x0001$	10 Takte

→ Wie kann man $x \cdot 2$ noch am Computer ausrechnen lassen?

→ Messen Sie die Rechenzeit für alle 3 Operationen

$0x0002 \cdot 0x0002$	22 Takte
$0xffff \cdot 0xffff$	22 Takte
$0x0002 \ll 0x0001$ (immediate)	10 Takte

4.) Division mit einer 2er-Potenz

analog: Siehe [Bit shift nach rechts]

Achtung: Hierbei entsteht niemals eine Gleitkommazahl (float, double)

Übung: → Was ist $5 \div 2$ auf dem ATmega328p? 2

→ Wieviele Taktzyklen benötigt

$0xffff / 0x0002$ **Aufgaben mit Pfeil -> potentielle Schulaufgabenaufgaben** 224 444

→ Wieviele Taktzyklen benötigt

$0xffff \,\%\, 0x0002$ 224 444 } Das ist katastrophal langsam ▽▽▽

→ Wie kann ich dieselben Rechnungen also

$x / 2$ bzw. $x \,\%\, 2$

noch ausführen? Wieviele Takte brauchen diese Instruktionen?

$x \gg 0x0001$ (immediate) 10

$x \,\&\, 0x0001$ 10

Zusammenfassung:

$x \cdot 1$	unnötig	$x / 1$	unnötig	$x \,\%\, 1$	unnötig immer 0
$x \cdot 2$	$\ll 0x1$	$x / 2$	$\gg 0x1$	$x \,\%\, 2$	$\& 0x1$
$x \cdot 4$	$\ll 0x2$	$x / 4$	$\gg 0x2$	$x \,\%\, 4$	$\& 0x3$
$x \cdot 8$	$\ll 0x3$	$x / 8$	$\gg 0x3$	$x \,\%\, 8$	$\& 0x7$
$x \cdot 16$	$\ll 0x4$	$x / 16$	$\gg 0x4$	$x \,\%\, 16$	$\& 0xf$
$x \cdot 32$	$\ll 0x5$	$x / 32$	$\gg 0x5$	$x \,\%\, 32$	$\& 0x1f$
$x \cdot 64$	$\ll 0x6$	$x / 64$	$\gg 0x6$	$x \,\%\, 64$	$\& 0x3f$
$x \cdot 128$	$\ll 0x7$	$x / 128$	$\gg 0x7$	$x \,\%\, 128$	$\& 0x7f$
$x \cdot 256$	$\ll 0x8$	$x / 256$	$\gg 0x8$	$x \,\%\, 256$	$\& 0xff$
$x \cdot 512$	$\ll 0x9$	$x / 512$	$\gg 0x9$	$x \,\%\, 512$	$\& 0x1ff$
$x \cdot 1024$	$\ll 0xa$	$x / 1024$	$\gg 0xa$	$x \,\%\, 1024$	$\& 0x3ff$
⋮		⋮		⋮	

x / 2er Potenz immediate und $x \,\%$ 2er Potenz immediate } wird vom Compiler optimiert 8

Übung: → vgl $x \,\%\, 0x0100$ mit $x \,\%\, 0xea3$

0x1010 % 0x8

0001 0000 0001 0000 % 1000 =

& . . . 0000111

----------0----------01

9 Takte vs 208 ▽ Achtung

$0x01ab \,\%\, 0x0008$

0000:0001:1010:1011 % 0000:0000:0000:1000 = ——— 0 ——— 011

Dieselbe Wirkung kann durch folgende Operation erzielt werden

0000000110101011

& 0 ——— 0111 ← Bitmaske

. . . . 00000011

Wiederholung: Bitoperationen in C

① Not

A	~A
0	1
1	0

② And

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

③ Or

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

④ Xor

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

Achtung: Verwechseln Sie die Bitoperationen (~, &, |, ^) nicht mit den logischen Verknüpfungen (!, &&, ||)

In C ist jeder Ausdruck, der nicht Null ist true

Ausdrücke die Null sind sind false

7||3 ⟶ true || true ⟶ true ⟶ 0x1

7|3 ⟶ 0111 | 0011 ⟶ 0111

```
uint8_t a = 7;
if ( a == 0 ) { . . }
```

false

if A else B;

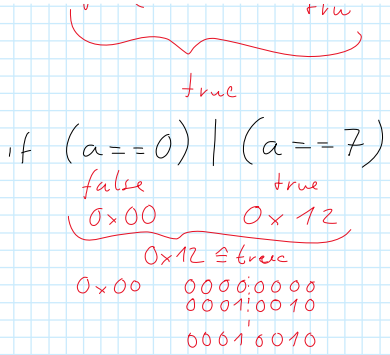
false ist 0x00

true sonst

```
uint8_t a = 7
if ( a == 0 ) || ( a == 7 )
```

A

false tru


true
if (a==0) | (a==7)
false
true
0x00
0x12
0x12 ≙ true
0x00
0000 0000
0001 0010
0001 0010